Exercise 1: Control Structures

Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

- **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

[SQL Worksheet]*

```oracle_plsql
BEGIN
    FOR c IN (SELECT CustomerID, DOB FROM Customers) LOOP
        IF MONTHS_BETWEEN(SYSDATE, c.DOB) / 12 > 60 THEN
            UPDATE Loans
            SET InterestRate = InterestRate - 1
            WHERE CustomerID = c.CustomerID;
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Final Loans Table:');
    FOR l IN (SELECT * FROM Loans) LOOP
        DBMS_OUTPUT.PUT_LINE('LoanID: ' || l.LoanID ||
                             ', CustomerID: ' || l.CustomerID ||
                             ', LoanAmount: ' || l.LoanAmount ||
                             ', InterestRate: ' || l.InterestRate ||
                             ', StartDate: ' || TO_CHAR(l.StartDate, 'YYYY-MM-DD') ||
                             ', EndDate: ' || TO_CHAR(l.EndDate, 'YYYY-MM-DD'));
    END LOOP;

    COMMIT;
END;
```

```
Final Loans Table:
LoanID: 1, CustomerID: 1, LoanAmount: 5000, InterestRate: 5, StartDate: 2025-06-21, EndDate: 2030-06-21


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.015
```

Scenario 2: A customer can be promoted to VIP status based on their balance.

- **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

[SQL Worksheet]*

```sql
ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN
    FOR c IN (SELECT CustomerID, Balance FROM Customers) LOOP
        IF c.Balance > 10000 THEN
            UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = c.CustomerID;
        ELSE
            UPDATE Customers SET IsVIP = 'FALSE' WHERE CustomerID = c.CustomerID;
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Final Customers Table:');
    FOR cust IN (SELECT * FROM Customers) LOOP
        DBMS_OUTPUT.PUT_LINE('CustomerID: ' || cust.CustomerID ||
                             ', Name: ' || cust.Name ||
                             ', DOB: ' || TO_CHAR(cust.DOB, 'YYYY-MM-DD') ||
                             ', Balance: ' || cust.Balance ||
                             ', IsVIP: ' || cust.IsVIP);
    END LOOP;

    COMMIT;
END;
```

```
Final Customers Table:
CustomerID: 1, Name: John Doe, DOB: 1985-05-15, Balance: 1000, IsVIP: FALSE
CustomerID: 2, Name: Jane Smith, DOB: 1990-07-20, Balance: 1500, IsVIP: FALSE


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```

Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

- **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

[SQL Worksheet]*

```
DECLARE
    v_due_date Loans.EndDate%TYPE;
    v_name     Customers.Name%TYPE;
BEGIN
    FOR l IN (
        SELECT LoanID, CustomerID, EndDate
        FROM Loans
        WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30
    ) LOOP
        SELECT Name INTO v_name FROM Customers WHERE CustomerID = l.CustomerID;

        DBMS_OUTPUT.PUT_LINE('Reminder: Dear ' || v_name ||
                             ', your loan (Loan ID: ' || l.LoanID ||
                             ') is due on ' || TO_CHAR(l.EndDate, 'YYYY-MM-DD'));
    END LOOP;
END;
```

No output because no customers found due in 30 days

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.019
```

Exercise 2: Error Handling

Scenario 1: Handle exceptions during fund transfers between accounts.

- **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
    p_from_account_id IN NUMBER,
    p_to_account_id   IN NUMBER,
    p_amount          IN NUMBER
) IS
    v_from_balance NUMBER;
BEGIN
    SELECT Balance INTO v_from_balance
    FROM Accounts
    WHERE AccountID = p_from_account_id;

    IF v_from_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account');
    END IF;

    UPDATE Accounts
    SET Balance = Balance - p_amount
    WHERE AccountID = p_from_account_id;

    UPDATE Accounts
    SET Balance = Balance + p_amount
    WHERE AccountID = p_to_account_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Transfer successful.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error during fund transfer: ' || SQLERRM);
END;
/
SET SERVEROUTPUT ON;
EXEC SafeTransferFunds(1, 2, 200);
```

```
SQL> EXEC SafeTransferFunds(1, 2, 200)


Transfer successful.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.077
```

Scenario 2: Manage errors when updating employee salaries.

- **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

[SQL Worksheet]*

```sql
CREATE OR REPLACE PROCEDURE UpdateSalary(
    p_employee_id IN NUMBER,
    p_percentage  IN NUMBER
) IS
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * p_percentage / 100)
    WHERE EmployeeID = p_employee_id;

    IF SQL%ROWCOUNT = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Employee ID not found');
    END IF;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Salary updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error updating salary: ' || SQLERRM);
END;

EXEC UpdateSalary(1, 10);
```

```
SQL> EXEC UpdateSalary(1, 10)


Salary updated successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```

Scenario 3: Ensure data integrity when adding a new customer.

- **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE AddNewCustomer(
    p_customer_id IN NUMBER,
    p_name        IN VARCHAR2,
    p_dob         IN DATE,
    p_balance     IN NUMBER
) IS
BEGIN
    INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified)
    VALUES(p_customer_id, p_name, p_dob, p_balance, SYSDATE);

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Customer added successfully.');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Customer ID already exists.');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Unexpected error adding customer: ' || SQLERRM);
END;

EXEC AddNewCustomer(3, 'David Green', TO_DATE('1980-01-10', 'YYYY-MM-DD'), 2000);
```

```
SQL> EXEC AddNewCustomer(3, 'David Green', TO_DATE('1980-01-10', 'YYYY-MM-DD'), 2000)


Customer added successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Exercise 3: Stored Procedures

Scenario 1: The bank needs to process monthly interest for all savings accounts.

- **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
    -- Apply 1% interest to all Savings accounts
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01)
    WHERE AccountType = 'Savings';

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Monthly interest applied to all Savings accounts.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error processing monthly interest: ' || SQLERRM);
END;
/

EXEC ProcessMonthlyInterest;
```

```
SQL> EXEC ProcessMonthlyInterest


Monthly interest applied to all Savings accounts.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.006
```

Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

- **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
    p_department IN VARCHAR2,
    p_bonus_pct  IN NUMBER  -- Pass 10 for 10%, etc.
) IS
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * p_bonus_pct / 100)
    WHERE Department = p_department;

    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No employees found in department ' || p_department);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Bonus applied to ' || SQL%ROWCOUNT || ' employees in ' || p_department);
    END IF;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error updating bonuses: ' || SQLERRM);
END;
/

EXEC UpdateEmployeeBonus('IT', 10);
```

```
SQL> EXEC UpdateEmployeeBonus('IT', 10)


Bonus applied to 1 employees in IT


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.009
```

Scenario 3: Customers should be able to transfer funds between their accounts.

- **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PROCEDURE TransferFunds(
    p_from_account_id IN NUMBER,
    p_to_account_id   IN NUMBER,
    p_amount          IN NUMBER
) IS
    v_balance NUMBER;
BEGIN
    -- Check if from-account has sufficient balance
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account_id;

    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
    END IF;

    -- Perform transfer
    UPDATE Accounts
    SET Balance = Balance - p_amount
    WHERE AccountID = p_from_account_id;

    UPDATE Accounts
    SET Balance = Balance + p_amount
    WHERE AccountID = p_to_account_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Transferred ' || p_amount || ' from Account ' ||
                         p_from_account_id || ' to Account ' || p_to_account_id);
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/

EXEC TransferFunds(1, 2, 500);
```

```
SQL> EXEC TransferFunds(1, 2, 500)


Transferred 500 from Account 1 to Account 2


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.017
```

Exercise 4: Functions

Scenario 1: Calculate the age of customers for eligibility checks.

- **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

[SQL Worksheet]*

```sql
CREATE OR REPLACE FUNCTION CalculateAge(p_dob DATE)
RETURN NUMBER IS
    v_age NUMBER;
BEGIN
    v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
    RETURN v_age;
END;
/

SELECT Name, CalculateAge(DOB) AS Age FROM Customers;
```

	NAME	AGE
1	John Doe	40
2	Jane Smith	34
3	David Green	45

Scenario 2: The bank needs to compute the monthly installment for a loan.

- **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
    p_loan_amount    IN NUMBER,
    p_interest_rate  IN NUMBER,
    p_duration_years IN NUMBER
)
RETURN NUMBER IS
    v_monthly_rate NUMBER;
    v_months       NUMBER;
    v_installment  NUMBER;
BEGIN
    v_monthly_rate := p_interest_rate / 12 / 100;
    v_months := p_duration_years * 12;

    IF v_monthly_rate = 0 THEN
        v_installment := p_loan_amount / v_months;
    ELSE
        v_installment := p_loan_amount * v_monthly_rate * POWER(1 + v_monthly_rate, v_months)
                         / (POWER(1 + v_monthly_rate, v_months) - 1);
    END IF;

    RETURN ROUND(v_installment, 2);
END;
/

SELECT CalculateMonthlyInstallment(50000, 5, 5) AS EMI FROM dual;
```

	EMI
1	943.56

Scenario 3: Check if a customer has sufficient balance before making a transaction.

- **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

[SQL Worksheet]*

```sql
CREATE OR REPLACE FUNCTION HasSufficientBalance(
    p_account_id IN NUMBER,
    p_amount     IN NUMBER
)
RETURN BOOLEAN IS
    v_balance NUMBER;
BEGIN
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_account_id;

    RETURN v_balance >= p_amount;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
    WHEN OTHERS THEN
        RETURN FALSE;
END;
/

DECLARE
    result BOOLEAN;
BEGIN
    result := HasSufficientBalance(1, 1000);
    IF result THEN
        DBMS_OUTPUT.PUT_LINE('Sufficient balance.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Insufficient balance.');
    END IF;
END;
```

```
SQL> DECLARE
         result BOOLEAN;
     BEGIN
         result := HasSufficientBalance(1, 1000);...
Show more...


Insufficient balance.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Exercise 5: Triggers

Scenario 1: Automatically update the last modified date when a customer's record is updated.

- **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

[SQL Worksheet]*

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
    :NEW.LastModified := SYSDATE;
END;
/

UPDATE Customers SET Balance = Balance + 100 WHERE CustomerID = 1;
SELECT LastModified FROM Customers WHERE CustomerID = 1;
```

	LASTMODIFIED
1	6/21/2025, 7:58:31

Scenario 2: Maintain an audit log for all transactions.

- **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

[SQL Worksheet]*

```sql
CREATE TABLE AuditLog (
    AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    TransactionID NUMBER,
    AccountID NUMBER,
    Amount NUMBER,
    TransactionType VARCHAR2(10),
    TransactionDate DATE,
    LoggedAt DATE DEFAULT SYSDATE
);

CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
    INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionType, TransactionDate)
    VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType, :NEW.TransactionDate);
END;
/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (3, 1, SYSDATE, 100, 'Deposit');

SELECT * FROM AuditLog;
```

	AUDITID	TRANSACTIONID	ACCOUNTID	AMOUNT	TRANSACTIONTYPI	TRANSACTIONDATI	LOGGEDAT
1	1	3	1	100	Deposit	6/21/2025, 7:59:35	6/21/2025, 7:59:35

Scenario 3: Enforce business rules on deposits and withdrawals.

- **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
    v_balance NUMBER;
BEGIN
    SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

    IF :NEW.TransactionType = 'Withdrawal' THEN
        IF :NEW.Amount > v_balance THEN
            RAISE_APPLICATION_ERROR(-20001, 'Withdrawal exceeds available balance.');
        END IF;
    ELSIF :NEW.TransactionType = 'Deposit' THEN
        IF :NEW.Amount <= 0 THEN
            RAISE_APPLICATION_ERROR(-20002, 'Deposit amount must be positive.');
        END IF;
    ELSE
        RAISE_APPLICATION_ERROR(-20003, 'Invalid transaction type.');
    END IF;
END;
/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (4, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (5, 1, SYSDATE, 99999, 'Withdrawal'); -- Should raise error
```

```
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
     VALUES (4, 1, SYSDATE, 200, 'Deposit')

1 row inserted.

Elapsed: 00:00:00.010
```

```
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
     VALUES (5, 1, SYSDATE, 99999, 'Withdrawal')

ORA-20001: Withdrawal exceeds available balance.
ORA-06512: at "SQL_UPMXLQRZGJHR231RVVFH1JDAYT.CHECKTRANSACTIONRULES", line 8
ORA-04088: error during execution of trigger 'SQL_UPMXLQRZGJHR231RVVFH1JDAYT.CHECKTRANSACTIONRULES'

https://docs.oracle.com/error-help/db/ora-20001/
Error at Line: 7 Column: 0
```

Exercise 6: Cursors

Scenario 1: Generate monthly statements for all customers.

- **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

[SQL Worksheet]*

```oracle_plsql
SET SERVEROUTPUT ON;
DECLARE
    CURSOR cur_transactions IS
        SELECT t.TransactionID, t.AccountID, a.CustomerID, t.TransactionDate, t.Amount, t.TransactionType
        FROM Transactions t
        JOIN Accounts a ON t.AccountID = a.AccountID
        WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')
        ORDER BY a.CustomerID;
    v_transaction_id Transactions.TransactionID%TYPE;
    v_account_id     Transactions.AccountID%TYPE;
    v_cust_id        Accounts.CustomerID%TYPE;
    v_date           Transactions.TransactionDate%TYPE;
    v_amount         Transactions.Amount%TYPE;
    v_type           Transactions.TransactionType%TYPE;
    v_prev_cust_id   Customers.CustomerID%TYPE := NULL;
    v_name           Customers.Name%TYPE;
BEGIN
    DBMS_OUTPUT.PUT_LINE('--- Monthly Statement ---');
    OPEN cur_transactions;
    LOOP
        FETCH cur_transactions INTO
            v_transaction_id, v_account_id, v_cust_id, v_date, v_amount, v_type;
        EXIT WHEN cur_transactions%NOTFOUND;
        IF v_prev_cust_id IS NULL OR v_cust_id != v_prev_cust_id THEN
            SELECT Name INTO v_name FROM Customers WHERE CustomerID = v_cust_id;
            DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Customer: ' || v_name || ' (ID: ' || v_cust_id || ')');
            DBMS_OUTPUT.PUT_LINE('-------------------------------------');
            v_prev_cust_id := v_cust_id;
        END IF;
        DBMS_OUTPUT.PUT_LINE(
            'TxnID: ' || v_transaction_id ||
            ', Account: ' || v_account_id ||
            ', Date: ' || TO_CHAR(v_date, 'YYYY-MM-DD') ||
            ', Type: ' || v_type ||
            ', Amount: $' || v_amount
        );
    END LOOP;
    CLOSE cur_transactions;
END;
/
```

```
SQL> DECLARE
        CURSOR cur_transactions IS
            SELECT t.TransactionID, t.AccountID, a.CustomerID, t.TransactionDate, t.Amount, t.TransactionType
            FROM Transactions t...
Show more...


--- Monthly Statement ---

Customer: John Doe (ID: 1)
-------------------------------------
TxnID: 3, Account: 1, Date: 2025-06-21, Type: Deposit, Amount: $100
TxnID: 4, Account: 1, Date: 2025-06-21, Type: Deposit, Amount: $200
TxnID: 1, Account: 1, Date: 2025-06-21, Type: Deposit, Amount: $200

Customer: Jane Smith (ID: 2)
-------------------------------------
TxnID: 2, Account: 2, Date: 2025-06-21, Type: Withdrawal, Amount: $300


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.017
```

Scenario 2: Apply annual fee to all accounts.

- **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

[SQL Worksheet]*

```
DECLARE
    CURSOR cur_accounts IS
        SELECT AccountID, Balance FROM Accounts;

    v_account_id Accounts.AccountID%TYPE;
    v_balance    Accounts.Balance%TYPE;
    v_fee        CONSTANT NUMBER := 100; -- Flat $100 annual fee
BEGIN
    OPEN cur_accounts;
    LOOP
        FETCH cur_accounts INTO v_account_id, v_balance;
        EXIT WHEN cur_accounts%NOTFOUND;

        IF v_balance >= v_fee THEN
            UPDATE Accounts
            SET Balance = Balance - v_fee,
                LastModified = SYSDATE
            WHERE AccountID = v_account_id;

            DBMS_OUTPUT.PUT_LINE('Annual fee applied to Account ID: ' || v_account_id);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Skipped Account ID ' || v_account_id || ': insufficient balance.');
        END IF;
    END LOOP;

    CLOSE cur_accounts;
    COMMIT;
END;
/
```

```
SQL> DECLARE
         CURSOR cur_accounts IS
             SELECT AccountID, Balance FROM Accounts;
     ...
Show more...


Annual fee applied to Account ID: 1
Annual fee applied to Account ID: 2


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.008
```

Scenario 3: Update the interest rate for all loans based on a new policy.

- **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

[SQL Worksheet]*

```
DECLARE
    CURSOR cur_loans IS
        SELECT LoanID, LoanAmount FROM Loans;

    v_loan_id    Loans.LoanID%TYPE;
    v_amount     Loans.LoanAmount%TYPE;
    v_new_rate   NUMBER;
BEGIN
    OPEN cur_loans;
    LOOP
        FETCH cur_loans INTO v_loan_id, v_amount;
        EXIT WHEN cur_loans%NOTFOUND;

        IF v_amount < 10000 THEN
            v_new_rate := 4;
        ELSIF v_amount <= 50000 THEN
            v_new_rate := 5;
        ELSE
            v_new_rate := 6;
        END IF;

        UPDATE Loans
        SET InterestRate = v_new_rate
        WHERE LoanID = v_loan_id;

        DBMS_OUTPUT.PUT_LINE('Loan ID ' || v_loan_id || ' updated to Interest Rate: ' || v_new_rate || '%');
    END LOOP;

    CLOSE cur_loans;
    COMMIT;
END;
/
```

```
SQL> DECLARE
         CURSOR cur_loans IS
             SELECT LoanID, LoanAmount FROM Loans;
     ...
Show more...


Loan ID 1 updated to Interest Rate: 4%


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

Exercise 7: Packages

Scenario 1: Group all customer-related procedures and functions into a package.

- **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

[SQL Worksheet]*

```oracle_sql
CREATE OR REPLACE PACKAGE CustomerManagement IS
    PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE, p_balance NUMBER);
    PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE, p_balance NUMBER);
    FUNCTION GetCustomerBalance(p_id NUMBER) RETURN NUMBER;
END CustomerManagement;
/
CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

    PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE, p_balance NUMBER) IS
    BEGIN
        INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
        VALUES (p_id, p_name, p_dob, p_balance, SYSDATE);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Customer ID already exists.');
    END;

    PROCEDURE UpdateCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE, p_balance NUMBER) IS
    BEGIN
        UPDATE Customers
        SET Name = p_name, DOB = p_dob, Balance = p_balance, LastModified = SYSDATE
        WHERE CustomerID = p_id;
    END;

    FUNCTION GetCustomerBalance(p_id NUMBER) RETURN NUMBER IS
        v_balance NUMBER;
    BEGIN
        SELECT Balance INTO v_balance FROM Customers WHERE CustomerID = p_id;
        RETURN v_balance;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
    END;

END CustomerManagement;
/
EXEC CustomerManagement.AddCustomer(3, 'David Green', TO_DATE('1980-01-10','YYYY-MM-DD'), 5000);
SELECT CustomerManagement.GetCustomerBalance(3) FROM dual;
```

	CUSTOMERMANAGEMENT.GETCUSTOMERBALANCE(3)
1	2000

Scenario 2: Create a package to manage employee data.

- **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

[SQL Worksheet]*

```sql
CREATE OR REPLACE PACKAGE EmployeeManagement IS
    PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_position VARCHAR2, p_salary NUMBER, p_dept VARCHAR2, p_hire_date DATE);
    PROCEDURE UpdateEmployee(p_id NUMBER, p_salary NUMBER);
    FUNCTION CalculateAnnualSalary(p_id NUMBER) RETURN NUMBER;
END EmployeeManagement;
/
CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

    PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_position VARCHAR2, p_salary NUMBER, p_dept VARCHAR2, p_hire_date DATE) IS
    BEGIN
        INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
        VALUES (p_id, p_name, p_position, p_salary, p_dept, p_hire_date);
    END;

    PROCEDURE UpdateEmployee(p_id NUMBER, p_salary NUMBER) IS
    BEGIN
        UPDATE Employees
        SET Salary = p_salary
        WHERE EmployeeID = p_id;
    END;

    FUNCTION CalculateAnnualSalary(p_id NUMBER) RETURN NUMBER IS
        v_salary NUMBER;
    BEGIN
        SELECT Salary INTO v_salary FROM Employees WHERE EmployeeID = p_id;
        RETURN v_salary * 12;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
    END;

END EmployeeManagement;
/
EXEC EmployeeManagement.HireEmployee(3, 'Carol White', 'Analyst', 40000, 'Finance', TO_DATE('2022-05-01','YYYY-MM-DD'));
SELECT EmployeeManagement.CalculateAnnualSalary(3) FROM dual;
```

	EMPLOYEEMANAGEMENT.CALCULATEANNUALSALARY(3)
1	480000

Scenario 3: Group all account-related operations into a package.

- **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

[SQL Worksheet]*

```oracle_plsql
CREATE OR REPLACE PACKAGE AccountOperations IS
    PROCEDURE OpenAccount(p_id NUMBER, p_cust_id NUMBER, p_type VARCHAR2, p_balance NUMBER);
    PROCEDURE CloseAccount(p_id NUMBER);
    FUNCTION GetTotalBalance(p_cust_id NUMBER) RETURN NUMBER;
END AccountOperations;
/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

    PROCEDURE OpenAccount(p_id NUMBER, p_cust_id NUMBER, p_type VARCHAR2, p_balance NUMBER) IS
    BEGIN
        INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
        VALUES (p_id, p_cust_id, p_type, p_balance, SYSDATE);
    END;

    PROCEDURE CloseAccount(p_id NUMBER) IS
    BEGIN
        DELETE FROM Accounts WHERE AccountID = p_id;
    END;

    FUNCTION GetTotalBalance(p_cust_id NUMBER) RETURN NUMBER IS
        v_total NUMBER;
    BEGIN
        SELECT SUM(Balance) INTO v_total
        FROM Accounts
        WHERE CustomerID = p_cust_id;
        RETURN NVL(v_total, 0);
    END;

END AccountOperations;
/
EXEC AccountOperations.OpenAccount(3, 1, 'Savings', 2000);
SELECT AccountOperations.GetTotalBalance(1) FROM dual;
```

```
SQL> EXEC AccountOperations.OpenAccount(3, 1, 'Savings', 2000)


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013
```